МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ВИСОКОШВИДКІСНІ ОБЧИСЛЮВАЛЬНІ СИСТЕМИ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для студентів, які навчаються за спеціальністю 121 «Інженерія програмного забезпечення»

Київ
КПІ ім. Ігоря Сікорського
2018

Сучасні високошвидкісні обчислювальні системи: комп'ютерний практикум [Електронний ресурс] : навч. посіб. для студ. спеціальності 121 «Інженерія програмного забезпечення» / КПІ ім. Ігоря Сікорського ; уклад.: О. С. Шкурат, В. Я. Юрчишин. – Електронні текстові дані (1 файл: 1,6 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 50 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 9 від 24.05.2018 р.)
за поданням Вченої ради факультету прикладної математики (протокол № 9 від 23.04.2018 р.)

Електронне мережне навчальне видання

СУЧАСНІ ВИСОКОШВИДКІСНІ ОБЧИСЛЮВАЛЬНІ СИСТЕМИ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ

Укладачі: *Шкурат Оксана Сергіївна*
Юрчишин Василь Якович, канд. техн. наук, доц.

Відповідальний редактор *Заболотня Тетяна Миколаївна,* канд. техн. наук, доц.

Рецензенти *Тимошин Юрій Афанасійович*, канд. техн. наук, доц.
Корнійчук Віктор Іванович, канд. техн. наук, доц.

Навчальний посібник розроблено для ознайомлення студентів з теоретичними відомостями та практичними прийомами роботи із сучасними високошвидкими обчислювальними системами та вимогами до виконання лабораторних робіт. Навчальне видання призначене для студентів, які навчаються за спеціальністю 121 Інженерія програмного забезпечення КПІ ім. Ігоря Сікорського.

ЗМІСТ

ВСТУП

Застосування інформаційних технологій на базі суперкомп'ютерних, кластерних та Grid-систем сприяють розробці сучасного програмного забезпечення, прикладом якого можуть виступати широкомасштабні системи моніторингу, управління та аналізу з глобально розподіленими джерелами даних.

У даному навчальному посібнику розглядаються теоретичні та практичні основи з використання технологій паралельних та розподілених обчислень, віртуалізації серверних систем, проектування корпоративних обчислювальних систем та застосування кластерних та гетерогенних розподілених обчислювальних систем із використанням мережевих протоколів взаємодії клієнтських та серверних додатків. Посібник складається з трьох розділів, кожен з яких присвячений виконанню певної лабораторної роботи з дисципліни «Сучасні високошвидкі обчислювальні системи», яка входить до циклу професійної підготовки «Навчальні дисципліни для здобуття універсальних компетентностей дослідника». В кожному розділі надаються теоретичні відомості відповідно до тематики роботи, вказівки щодо виконання завдання, а також наводяться варіанти завдань, вимоги до оформлення звіту з виконаної лабораторної роботи, контрольні питання для самоперевірки та список рекомендованої літератури.

Лабораторні роботи з дисципліни «Сучасні високошвидкі обчислювальні системи» розраховані на 18 академічних годин аудиторних занять.

ЛАБОРАТОРНА РОБОТА №1. Робота на кластері з використанням технології MPI

Мета роботи: Навчитись працювати з кластерами на базі *Linux* та *Windows HPC*.
Завдання: Створити програму відповідно до свого номеру варіанту, що для заданого у командному рядку діапазону цілих чисел та рядка знаходить всі числа з діапазону.

Теоретичні відомості та методичні вказівки

До початку виконання лабораторної роботи необхідно ознайомитись з інструкцією роботи кластеру НТУУ «КПІ ім. Ігоря Сікорського». Детальна інформація представлена у посиланні [1].

Кластерна обчислювальна система Центру суперкомп'ютерних обчислень НТУУ «КПІ ім. Ігоря Сікорського» працює під управлінням операційної системи *Linux*. В *Linux* стандартним засобом віддаленого доступу є протокол *Secure Shell* (скорочено *SSH*). Віддалений доступ можливий як через Інтернет, так і з мережі НТУУ «КПІ ім. Ігоря Сікорського». Найбільш популярні програми-клієнти, які дозволяють встановлювати зв'язок за протоколом *SSH* - це утиліти *ssh* в *Linux* та *PuTTY* для *Windows*.

Розглянемо роботу з програмою *PuTTY*, яку можна завантажити з сайту: http://www.putty.org/. Для запуску задач на кластерній системі необхідно мати на ній акаунт. Акаунт створюється адміністратором по заявці. Вам мають бути повідомлені наступні дані:

- логін вашого акаунта (далі *user*);
- пароль вашого акаунта (далі *pass*);
- IP-адресу кластеру або його символьне ім'я (далі *host*);
- порт, на який встановлюється *SSH*-з'єднання (далі *port*).

1. Віддалений доступ до командного рядка

1.1 Віконний варіант *PuTTY*

Після інсталяції програми *PuTTY* можна користуватись як віконним варіантом програми, так і консольним. Обидва варіанти представлені виконуваним файлом *putty.exe*. Запуск цього файлу без параметрів відкриває головне вікно програми: *Пуск → Програми → PuTTY → PuTTY*).

Віконна версія програми має перевагу в тому, що вона дозволяє зберігати «сесію» - адресу, порт серверу і тип підключення. Для створення сесії необхідно у головному вікні (рис. 1) ввести адресу сервера *host* в полі «*Host Name*», порт *port* в полі «*Port*» і обрати тип з'єднання *SSH*. В полі «*Saved Sessions*» необхідно ввести назву сесії (довільне ім'я) і натиснути кнопку «*Save*». Введене символьне ім'я з'явиться в списку нижче і відтепер завжди можна завантажити збережені параметри, обравши зі списку відповідне символьне ім'я і натиснувши кнопку «*Load*». Після завантаження параметрів для встановлення з'єднання з сервером необхідно натиснути кнопку «*Open*». Після цього має відкритись вікно консолі (рис. 2).

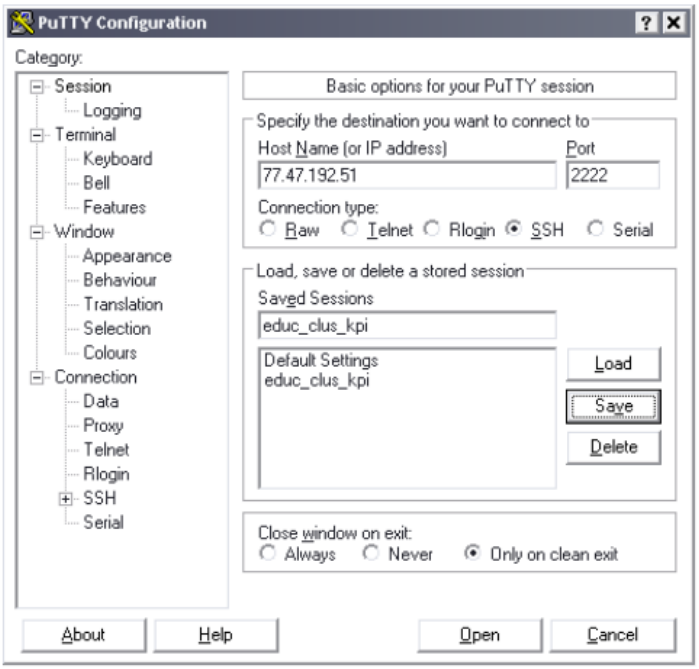

Рис. 1. Головне вікно *PuTTY*.

Рис. 2. Вікно консолі.

Якщо консоль не з'явилась, а натомість програма повідомила про помилку, то це означає, що або не працює мережа, або не працює сервер, або ви неправильно ввели надані вам дані. В даній консолі необхідно ввести ваш *user*, а потім *pass* (введені символи паролю не відображаються на екрані з метою забезпечення безпеки). Після цього ви отримаєте доступ до командного рядка, а поточним каталогом стане домашній каталог користувача.

1.2 Консольний варіант *PuTTY*

Розглянемо запуск консольного варіанту програми *PuTTY*: вибираємо *Пуск → Виконати*, набираємо в полі «*cmd*» і натискаємо клавішу *Enter*. Після цього з'явиться системна консоль (рис. 3), звідки можна запустити програму.

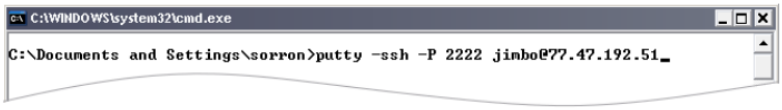

Рис. 3. Запуск *PuTTY* з командного рядка *Windows*.

Для встановлення з'єднання необхідно ввести командний рядок у наступному форматі:

```
C:\>putty -ssh -P port user@host
```

Після цього натискаємо *Enter* і з'являється вікно віддаленої консолі обчислювального вузла, де необхідно ввести пароль. Після вводу коректного пароля відбувається вхід у систему і перехід до домашньої директорії користувача:

```
login as: user
user@77.47.192.51's password:
Last login:
```

```
user@n001$ pwd
/home/user
user@n001$
```

Необхідно зауважити, що таким чином відкривається доступ до першого (головного) вузла кластеру. Для того, щоб перейти на інший вузол необхідно дати команду, в якій <*nodenumber*> – номер необхідного вузла.

```
user@n001$ ssh n<nodenumber>
```

Приклад переходу на вузол *n002*:

```
user@n001$ ssh n002
user@n002's password:
Last login:
user@n002$
```

2. Робота з файлами на кластері

Для повноцінної роботи з кластером необхідна також можливість завантажувати файли на кластер і з кластера. Для цього разом з програмою *PuTTY* встановлюється утиліта *pscp.exe* (*PuTTY secure copy client*).

Зверніть увагу! Програма *pscp.exe* працює тільки в консолі *Windows*. Необхідно виконати команду: *Пуск* → *Виконати*, набрати в полі «*cmd*» і натиснути клавішу *Enter*. З'явиться системна консоль, звідки можна запускати програму *pscp.exe*.

2.1 Перегляд списку файлів

Для виведення списку файлів в заданій директорії на обчислювальному вузлі призначена команда консолі *Windows*:

```
C:\>pscp -P 2222 -ls user@host:directory
```

Після вводу кожної команди відбувається запит пароля користувача. Приклад виконання команди:

```
C:\>pscp -P 2222 -ls user@77.47.192.51:/home/user
user@77.47.192.51's password:
Listing directory /home/user
drwx------ 4 user user 4096 Apr 21 20:14 .
drwxr-xr-x 83 root root 4096 Apr 26 13:22 ..
-rw------- 1 user user 432 Apr 27 06:46 .bash_history
-rw-r--r-- 1 user user 33 Apr 20 11:54 .bash_logout
-rw-r--r-- 1 user user 176 Apr 20 11:54 .bash_profile
-rw-r--r-- 1 user user 124 Apr 20 11:54 .bashrc
```

```
drwx------ 2 user user 4096 Apr 21 19:18 .ssh
-rw------- 1 user user 704 Apr 21 20:14 .viminfo
drwxr-xr-x 2 user user 4096 Apr 21 20:14 gmptest
C:\>
```

2.2 Копіювання файлів на кластер

Для копіювання одного файлу на обчислювальний вузол введіть команду в консолі *Windows*:

```
C:\>pscp -P port localfile user@host:file
```

В даній команді *localfile* – ім’я файлу на локальному комп’ютері, *file* – ім’я файлу на обчислювальному вузлі. Приклад виконання команди:

```
C:\>pscp -P 2222 C:\foo.txt user@77.47.192.51:/home/user/foo.txt
user@77.47.192.51’s password:
foo.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
C:\>
```

Перевіримо наявність файлу на обчислювальному вузлі командою *ls*:

```
user@n001$ ls
foo.txt gmptest
user@n001$
```

Також на кластер можна скопіювати каталог і всі його підкаталоги. Для цього слід додати параметр *-r*:

```
C:\>pscp -P port -r localdirectory user@host:directory
```

В даній команді *localdirectory* – ім’я директорії на локальному комп’ютері, яка буде скопійована, *directory* – ім’я директорії на сервері в яку буде скопійовано локальну директорію. Приклад:

```
C:\dir>dir
Содержимое папки C:\dir
27.04.2010 07:23 <DIR> .
27.04.2010 07:23 <DIR> ..
27.04.2010 07:23 46 foo1.txt
27.04.2010 07:23 <DIR> innerdir
1 файлов 46 байт
3 папок 1 773 776 896 байт свободно
C:\dir>cd innerdir
C:\dir\innerdir>dir
Содержимое папки C:\dir\innerdir
27.04.2010 07:23 <DIR> .
27.04.2010 07:23 <DIR> ..
27.04.2010 07:23 46 foo2.txt
1 файлов 46 байт
2 папок 1 773 776 896 байт свободно
C:\dir\innerdir>pscp -P 2222 -r C:\dir user@77.47.192.51:/home/user/
user@77.47.192.51’s password:
foo1.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
```

```
foo2.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
C:\dir\innerdir>
```

Перевіримо наявність каталогу на обчислювальному вузлі командою *ls:*

```
user@n001:~$ ls
dir foo.txt gmptest
user@n001:~/dir$ cd dir
user@n001$ ls
foo1.txt innerdir
user@n001:~/innerdir$ cd innerdir
user@n001:~/innerdir$ ls
foo2.txt
user@n001:~/innerdir$
```

2.3 Копіювання файлів з кластеру

Для копіювання файлів з обчислювального вузла на локальний комп’ютер призначена команда:

```
C:\>pscp -P port user@host:file localfile
```

Приклад роботи команди:

```
C:\dir>pscp -P 2222 user@77.47.192.51:/home/user/dir/innerdir/foo2.txt
C:\dir\foo3.txt
user@77.47.192.51’s password:
foo3.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
C:\dir>dir
Содержимое папки C:\dir
27.04.2010 07:23 <DIR> .
27.04.2010 07:23 <DIR> ..
27.04.2010 07:23 46 foo1.txt
27.04.2010 07:51 46 foo3.txt
27.04.2010 07:23 <DIR> innerdir
2 файлов 92 байт
3 папок 1 773 776 896 байт свободно
C:\dir>
```

3. Компіляція та запуск паралельних програм на кластері

3.1 Система управління чергою задач

3.1.1. Призначення системи управління чергою задач

Під час запуску програм на своєму персональному комп’ютері ви точно знаєте, скільки програм вже запущено та скільки вони використовують ресурсів комп’ютера. Виходячи з цієї інформації ви вирішуєте, коли і скільки програм запускати.

Запуск програм на кластері відрізняється від запуску програм на персональному комп’ютері: ви не знаєте, скільки та яких ресурсів

використовується в даний час на кожному з вузлів кластеру. Тому ви не можете самостійно обрати, на яких вузлах кластеру запускати свої програми. Для вирішення цієї проблеми використовується *система управління чергою задач*. Ідея такої системи полягає в наступному: всі користувачі кластеру повідомляють системі управління чергою, які програми вони бажають запустити та скільки обчислювальних ресурсів цим програмам потрібно. Заявки від користувачів кластеру кладуться в чергу. Система управління чергою самостійно запускає програми з черги тоді, коли може задовольнити потреби програми в ресурсах. Таким чином, система управління чергою має повну інформацію про те, які ресурси кластеру зайняті (та якими саме програмами яких користувачів), а які ресурси кластеру вільні. Виходячи зі сказаного вище, можна зробити висновок: для продуктивної роботи всіх користувачів кластеру необхідно (та в інтересах користувачів) запускати обчислювальні програми на кластері тільки через систему управління чергою задач.

Після того, як ви отримали доступ до командного рядка кластеру за допомогою протоколу *SSH*, ви можете виконувати команди на вузлі *n001*. На цьому вузлі дозволяється виконувати переміщення та копіювання даних, компіляцію програм, запуск обчислювальних програм за допомогою системи управління чергою. На інших вузлах дозволяється виконувати лише інформаційні команди під час визначення причин проблем роботи програм.

На кластері Центру суперкомп’ютерних обчислень НТУУ «КПІ ім. Ігоря Сікорського» встановлена реалізація системи управління чергою задач *PBS* (англ. *Portable Batch System*) *Torque*. Нижче розглянуті основні команди для роботи з *Torque*.

3.1.2 Команда *qsub*

Команда *qsub* призначена для додавання програми в чергу виконання. Після успішного виконання програма видає на консоль ідентифікатор задачі в системі управління чергами (не плутати з рангом задачі в *MPI* або ідентифікатором задачі в *OpenMP*). Цей ідентифікатор знадобиться для відстежування стану виконання програми або для видалення програми з черги.

```
qsub скрипт_старту_програми
```

скрипт_старту_програми – це ім’я файлу, в якому вказується деяка інформація для системи управління чергою задач та команди для запуску програми. Цей файл має наступний синтаксис:

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -N <назва задачі>
#PBS -l <специфікація ресурсів>
#PBS -o <ім’я файлу для виводу stdout>
#PBS -e <ім’я файлу для виводу stderr>
cd <каталог з програмою>
<команда запуску програми>
```

Призначення рядків цього файлу наступне:

-S /bin/bash – параметр вказує планувальнику, що команди запуску програми слід виконувати за допомогою оболонки /bin/bash (краще не змінюйте цей параметр).

-N <назва задачі> – параметр задає назву задачі. Назва буде відображатись у списку задач (необов’язковий параметр).

-l <специфікація ресурсів> – параметр задає скільки та яких ресурсів необхідно виділити програмі. Різні види ресурсів розділяються комою. Можна задавати наступні види ресурсів:

- *walltime=HH:MM:SS* – необхідний час виконання програми (HH - години, MM - хвилини, SS - секунди). Це реальний час виконання програми, а не машинний. Якщо програма буде виконуватись довше заявленого часу, вона буде примусово завершена.

- *nodes=N:ppn=P* – необхідна кількість вузлів (*N*) та ядер на кожному вузлі (*P*). Всього програмі буде виділено $N \times P$ ядер.
- Хоча в *Torque* можна задавати ресурси оперативної пам'яті, на кластері Центру суперкомп'ютерних обчислень НТУУ «КПІ ім. Ігоря Сікорського» оперативна пам'ять розподіляється за принципом: гарантований *1 Гб* на кожне обчислювальне ядро.

-o <ім'я файлу для виводу stdout> – повний шлях до файлу, в який буде записано стандартний потік виводу програми (при звичайному запуску вручну така інформація виводиться на консоль; необов'язковий параметр).

-e <ім'я файлу для виводу stderr> – повний шлях до файлу, в який буде записано стандартний потік помилок програми (при звичайному запуску вручну ця інформація виводиться на консоль; необов'язковий параметр).

Якщо не вказати імена файлів для запису виводу програми, то ці файли створюються у поточному каталозі, а імена файлів починаються з назви задачі.

3.1.3 Команда *qstat*

Команда *qstat* призначена для спостереження за станом виконання задач. Дану команду можна виконувати без параметрів (рис. 4):

```
user@n001$ qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
1000.pbs                  example.sh       user            01:11:12 R batch
1001.pbs                  example42.sh     user            01:15:01 R batch
1002.pbs                  run.sh           user                   0 Q batch
```

Рис. 4. Команда *qstat*.

В даному випадку видно, що користувач додав три задачі до черги. В стовпцях вказана наступна інформація:

- *Job id* – ідентифікатори задач;
- *Name* - ім'я скрипту старту програми або назва задачі (якщо задана).
- *Time Use* - кількість використаного процесорного часу.

- *S* - стан виконання задачі: «*Q*» - чекає в черзі, «*R*» - виконується.

Також команду *qstat* можна виконувати з параметром *-a* (виводить відносно більше інформації) чи з параметром *-f* (виводить дуже детальну інформацію).

3.1.4 Команда *qdel*

Команда *qdel* призначена для видалення задач із черги. Видаляти можна як задачу, яка очікує в черзі, так і задачу, яка вже виконується. В останньому випадку програма буде аварійно завершена.

```
user@n001$ qdel ідентифікатор_задачі
```

ідентифікатор_задачі – ідентифікатор задачі, який був виданий на консоль командою *qsub*. Якщо ви не пам’ятаєте цього ідентифікатору, дізнайтесь його командою *qstat*.

3.2 Компіляція та запуск програм на *MPI*

Нехай у вас є вихідні коди програми, написаної з використанням *MPI*. Для запуску цієї програми на кластері необхідно виконати наступні кроки:

1) Створіть окремий каталог для цієї програми. Наприклад, якщо ваш логін *user* та ви хочете створити каталог *example*, виконайте наступну команду:

```
user@n001$ mkdir /home/user/example
```

2) Завантажте у створений каталог вихідні коди програми.
3) Перейдіть у створений каталог командою *cd* (англ. *Change Directory*):

```
user@n001$ cd /home/user/example
```

4) Скомпілюйте програму. Якщо програма написана стороннім розробником, то разом із програмою можуть поставлятись інструкції для компіляції. Приклади програм на мові *C*, що написані з використанням *MPI*, компілюються наступним чином:

```
user@n001$ mpicc -W -Wall -O2 -std=c99 \
ім’я_файлу_1.c ім’я_файлу_2.c ... -o program
```

ім'я_файлу_1.c, ім'я_файлу_2.c – імена всіх файлів, що входять до складу програми; *program* – ім'я для виконуваного файлу програми, що створить компілятор. Якщо програма написана на мові C++, необхідно використовувати компілятор *mpic++*.

5) Створіть в каталозі програми скрипт запуску для системи управління задачами (чи створіть цей файл на вашому персональному комп'ютері та завантажте його на кластер у каталог програми). Нехай ім'я цього файлу буде */home/user/example/runjob.sh*. Вміст файлу може виглядати наступним чином:

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l nodes=7:ppn=2,walltime=02:00:00
#PBS -o /home/user/example/stdout.txt
#PBS -e /home/user/example/stderr.txt
cd /home/user/example
mpiexec ./program
```

Цей скрипт повідомляє *Torque* про те, що програмі необхідно виділити по 2 ядра на 7 вузлах, також програмі знадобиться якнайбільше 2 години часу для обчислень, вивід програми буде перенаправлений у файли */home/user/example/stdout.txt* та */home/user/example/stderr.txt*.

Зверніть увагу! Запускаючи *MPI* програми на кластері через систему управління чергою задач, не треба передавати команді *mpiexec* додаткових параметрів *-n* чи *-np*, що задають кількість задач. Ця інформація буде передана до *mpiexec* системою управління чергою задач автоматично.

6) Створіть або завантажте на кластер вхідні дані для програми.
7) Додайте програму в чергу виконання командою *qsub:*

```
user@n001$ qsub /home/user/example/runjob.sh
```

8) Стан виконання задачі можна дізнатись за допомогою команди *qstat.*

3.3 Компіляція та запуск програм на *OpenMP*

Нехай у вас є вихідні коди програми, написаної з використанням *OpenMP*. Для запуску цієї програми на кластері необхідно виконати наступні кроки:

1) Створіть окремий каталог для цієї програми. Наприклад, якщо ваш логін *user* та ви хочете створити каталог *example*, виконайте наступну команду:

```
user@n001$ mkdir /home/user/example
```

2) Завантажте у створений каталог вихідні коди програми.
3) Перейдіть у створений каталог командою *cd* (англ. *Change Directory*):

```
user@n001$ cd /home/user/example
```

4) Скомпілюйте програму. Якщо програма написана стороннім розробником, разом з програмою можуть поставлятись інструкції для компіляції. Приклади програм на мові *C* з даного документу, що написані з використанням *OpenMP*, компілюються наступним чином:

```
user@n001$ gcc -W -Wall -O2 -std=c99 -fopenmp \
ім'я_файлу_1.c ім'я_файлу_2.c ... -o program
```

ім'я_файлу_1.c, *ім'я_файлу_2.c* – імена всіх файлів, що входять до складу програми; *program* – ім'я для виконуваного файлу програми, що створить компілятор. Якщо програма написана на мові *C++*, необхідно використовувати компілятор *g++*.

5) Створіть в каталозі програми скрипт запуску для системи управління задачами (чи створіть цей файл на вашому персональному комп'ютері та завантажте його на кластер у каталог програми). Нехай ім'я цього файлу буде */home/user/example/runjob.sh*. Вміст файлу може виглядати наступним чином:

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l nodes=1:ppn=8,walltime=02:00:00
#PBS -o /home/user/example/stdout.txt
```

```
#PBS -e /home/user/example/stderr.txt
cd /home/user/example
export OMP_NUM_THREADS=8
./program
```

Цей скрипт повідомляє *Torque* про те, що програмі необхідно виділити 8 ядер на одному вузлі, програмі також знадобиться якнайбільше 2 години часу для обчислень, вивід програми буде перенаправлений у файли */home/user/example/stdout.txt* та */home/user/example/stderr.txt*. Значення змінної *OMP_NUM_THREADS* має бути рівне кількості ядер, вказаних в параметрі *ppn=P*.

Зверніть увагу! Запускаючи *OpenMP* програми на кластері через систему управління чергою задач, не слід виділяти програмі більше одного вузла, оскільки програми на *OpenMP* працюють тільки на системах зі спільною пам’яттю.

6) Створіть або завантажте на кластер вхідні дані для програми.
7) Додайте програму в чергу виконання командою *qsub*:

```
user@n001$ qsub /home/user/example/runjob.sh
```

8) Стан виконання задачі можна дізнатись за допомогою команди *qstat*.

3.4 Тестові запуски програм

Тестові (пробні для налагодження) запуски обчислювальних програм дозволяють виконувати задачі на інтерфейсному вузлі для уникнення очікування в черзі. Вимогою до тестових задач є використання не більше 4 ядер та час виконання – не більший 5 хвилин.

3.4.1 Тестовий запуск програм на *MPI*.

Виконайте завантаження програми та її вхідних даних на кластер, а також компіляцію програми. Після цього виконайте команду, де число 4 – це кількість задач:

```
user@n001$ mpiexec -np 4 ./program
```

Рекомендації до виконання

Розглянемо основи роботи з кластером на основі *Linux*. Для спрощення запуску паралельних задач *MPI* необхідне налаштування доступу за ключами між вузлами кластеру НТУУ КПІ ім. Ігоря Сікорського. Ключ *RSA* або *DSA* можна створити за допомогою утиліти *ssh-keygen*. Потрібно згенерувати ключі без захисту паролем, тому на запит «*Enter passphrase:*» слід просто натиснути *Enter*.

```
[test1@n001 ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/users/test1/.ssh/id_rsa):
Created directory '/home/users/test1/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/users/test1/.ssh/id_rsa.
Your public key has been saved in /home/users/test1/.ssh/id_rsa.pub.
The key fingerprint is:
36:3c:90:53:b6:ff:19:23:fc:58:0e:a2:17:75:07:c6 test1@n001
```

Після створення ключів, потрібно додати їх у список довірених:

```
[test1@n001 ~]$ cat .ssh/id_rsa.pub > .ssh/authorized_keys
```

Права доступу файлів в каталозі *.ssh/* контролюються при авторизації користувача та мають бути:

```
drwx------ user usergroup .ssh/
-rw-r--r-- user usergroup known_hosts
-rw-r--r-- user usergroup id_rsa.pub
-rw------- user usergroup id_rsa
-rw-r--r-- user usergroup authorized_keys
```

Після формування списку довірених ключів необхідно створити файл *known_hosts*. Можна скористатись готовим файлом */home/users/known_hosts*, який потрібно скопіювати у директорію *.ssh/*, що у домашньому каталозі користувача, або згенерувати самостійно за допомогою наступної команди:

```
[test1@n001 ~]$ for i in `seq -w 1 112`; do ssh -o StrictHostKeyChecking=no
n$i.kpi hostname; done;
Warning: Permanently added 'n001.kpi,172.16.1.1' (RSA) to the list of known
hosts.
…
```

Тестування отриманої конфігурації:

```
$ mpicc test.c -o test
$ mpirun -np 512 -H $(seq -f n%03g 112) test
```

Розглянемо основи роботи з кластером на основі *Windows HPC Server*. Доступ відбувається за допомогою утиліти *mstsc.exe* (*Remote Desktop client*) (рис. 5, 6).

Рис. 5. Робота з кластером.

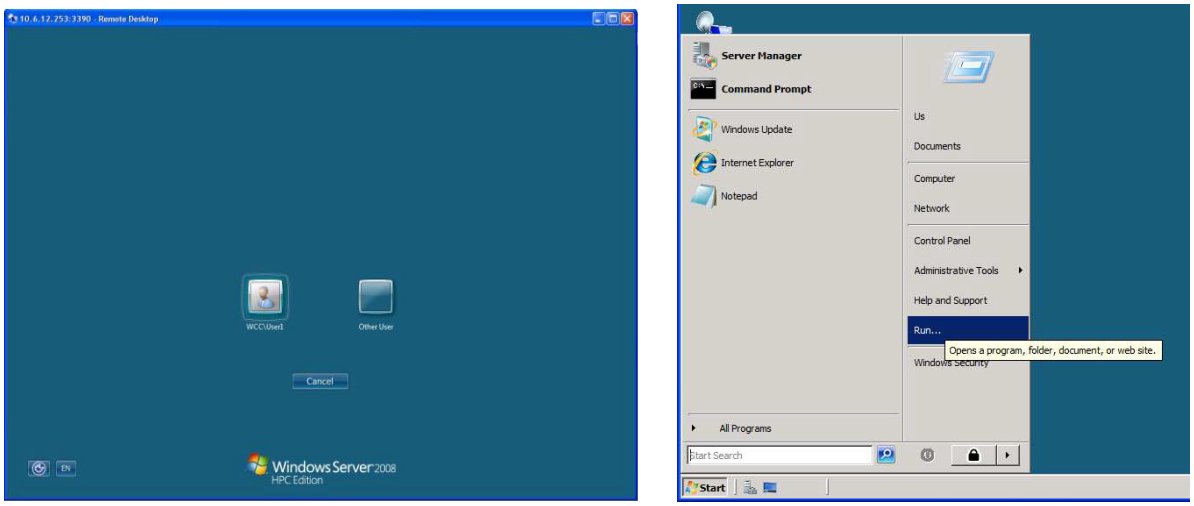

Рис. 6. Робота з кластером на основі *Windows HPC Server*.

Логін та пароль користувача надаються викладачем. Реалізація *MS MPI* та середовище розробки встановлюється на персональні комп'ютерні лабораторії.

Завдання

Реалізуйте програму, яка для заданого у командному рядку діапазону цілих чисел та рядка *X* знаходить всі числа з діапазону, для яких *SHA-256 геш* від текстового запису числа починається на *X*. Наприклад, якщо задано

діапазон [1,1000] та *X*=73475cb40a56, програма повинна знайти число 42, оскільки:

```
$ echo -n 42 | sha256sum | cut -c 1-12
73475cb40a56
```

При реалізації використовуйте *MPI* та тестові ресурси кластерів НТУУ «КПІ ім. Ігоря Сікорського».

Варіанти завдань

Діапазон цілих чисел: [1,10000000].

Варіант	**Значення рядка *X***
1	X = 25449d48ed84
2	X = 75e698f33b1d
3	X = 237d5ddbe920
4	X = 78e5f7e38d76
5	X = 818a2a1a2efe
6	X = d7187cd46d9b
7	X = 45c8ad3440d2
8	X = 0d50c4d6882d
9	X = d287df64b232
10	X = 73b881da57e1
11	X = 8a8ac1332386
12	X = 44d886879bb4
13	X = 200c2356535a
14	X = eb11eb72ef53
15	X = be0573b6fe9b
16	X = f4475869f4e7
17	X = 6584585f2b30
18	X = 3b5ab9e8a187

19	X = 03c8fb6367a4
20	X = 1a127fdeba72

Вимоги до оформлення звіту

Звіт має включати:

1. Титульний аркуш.
2. Індивідуальне завдання на лабораторну роботу.
3. Код програмної реалізації завдання.
4. Результати роботи.

Контрольні запитання

1. Для чого використовується протокол SSH?
2. Які функції виконує система управління чергою задач?
3. Для чого використовується команда *qsub* у системі управління чергою задач?
4. Команда *qstat*. Назвіть призначення та параметри команди.
5. Для чого використовується команда *qdel*?

Рекомендована література

1. Засоби паралельного програмування [Текст] / С.Г. Стіренко, Д.В. Грибенко, О.І. Зіненко, А.В. Михайленко – К., 2011. – 154 с.
2. Інструкція по використанню кластеру НТУУ «КПІ ім. Ігоря Сікорського» [Електронний ресурс]. – Режим доступу: http://hpcc.kpi.ua/uk/.

ЛАБОРАТОРНА РОБОТА №2. Створення віртуальних машин

Мети роботи: Навчитись працювати з технологіями віртуалізації *Hyper-V* та *VMware* для створення віртуальних машин і реалізації на їх основі сучасних високошвидкісних обчислень.

Завдання: Налаштувати віртуальну машину, використовуючи технології *Microsoft Hyper-V* та *VMware Workstation*.

Теоретичні відомості та методичні вказівки

На одному звичайному комп'ютері, навіть дуже потужному не може бути запущено декілька операційних систем одночасно. Але за допомогою спеціального програмного забезпечення на комп'ютері можна створити як би ще один віртуальний комп'ютер (або декілька) зі своїм віртуальним «залізом», на якому можна запустити іншу операційну систему, яка буде керувати віртуальним комп'ютером як реальним. В підсумку матимемо дві (або декілька) одночасно запущені операційні системи.

Вся процедура буде виглядати наступним чином:

- Завантажити комп'ютер в наявній операційній системі, яка буде основною. В ній встановлюєте спеціальну програму – програму віртуальних машин.
- В завантаженій операційній системі запускаєте програму віртуальних машин. За допомогою цієї програми створюєте віртуальну машину під певну операційну систему, яка буде запускатись в якості другорядної, і задаєте всі параметри віртуального комп'ютера: розмір жорсткого диска (цей розмір виділиться на реальному жорсткому диску в вигляді окремого файлу), об'єм оперативної пам'яті тощо.

– В вікні віртуальної машини запускаєте створену віртуальну машину. Встановлюєте на ній операційну систему, ніби як на реальному комп’ютері (тільки ці дії відбуваються в окремому вікні). Після установки операційної системи вона буде готова до роботи і буде автоматично запускатися в вікні цієї віртуальної машини.

Таким чином можна запустити декілька віртуальних машин із кількома операційними системами.

Взагалі можна на запущену віртуальну машину встановити ще одну віртуальну машину і на ній в свою чергу запустити вкладену операційну систему.

1. Microsoft Hyper-V

Microsoft Hyper-V являє собою рішення для віртуалізації серверів на базі процесорів з архітектурою *x64*. *Hyper-V* являється вбудованим компонентом 64-розрядних версій *Windows Server 2008 Standard*, *Windows Server 2008 Enterprise* та *Windows Server 2008 Datacenter*.

Дана технологія недоступна в 32-р*озрядних версіях Windows Server 2008*, в *Windows Server 2008 Standard*, *Windows Server 2008 Enterprise*, *Windows Server 2008 Datacenter* без *Hyper-V*, в *Windows Web Server 2008* та *Windows Server 2008* для систем на базі *Itanium*.

Hyper-V являє собою гіпервізор, тобто деякий прошарок між обладнанням і віртуальними машинами за рівнем нижче операційної системи. Ця архітектура була спочатку розроблена *IBM* в 1960-ті роки для мейнфреймів (великих *EOM*) і недавно стала доступною на платформах *x86/x64* як частина ряду рішень, включаючи *Windows Server 2008 Hyper-V* та *Vmware ESX*.

Архітектура віртуалізації з гіпервізором показана на рис. 7. Віртуалізація на базі гіпервізора основана на тому, що між обладнанням і

віртуальними машинами появляється прошарок, який перехоплює звернення операційних систем до процесора, пам’яті та інших пристроїв. При цьому доступ до периферійних пристроїв в різних реалізаціях гіпервізорів може бути організований по-різному. З точки зору існуючих рішень для реалізації менеджера віртуальних машин можна виділити два основних види архітектури гіпервізора: мікроядерну та монолітну.

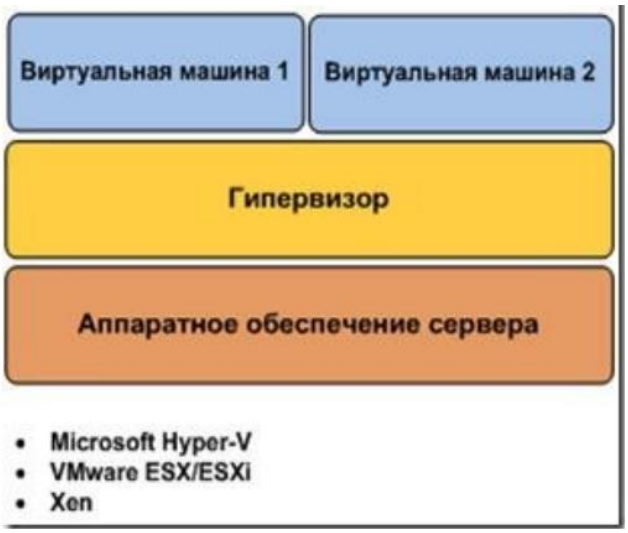

Рис. 7. Реалізація віртуалізації.

Монолітний підхід припускає, що всі драйвери пристроїв поміщені в гіпервізор. В монолітній моделі для доступу до обладнання гіпервізор використовує власні драйвери. Гостьові операційні системи працюють на віртуальних машинах поверх гіпервізора. Коли гостьовій системі потрібен доступ до обладнання, вона має пройти через гіпервізор та його модель драйверів. Зазвичай одна із гостьових операційних систем грає роль адміністратора чи консолі, в котрій запускаються компоненти для представлення ресурсів, управління та моніторингу всіх гостьових операційних систем.

В моделі монолітного гіпервізора існує проблема обновлення драйверів. При необхідності обновлення драйвера деякого пристрою (наприклад, мережевого адаптера) це можна зробити тільки разом з

виходом нової версії гіпервізора, в яку буде інтегрований новий драйвер для даного пристрою.

Мікроядерний підхід використовує дуже тонкий, спеціалізований гіпервізор, що виконує лише основні задачі забезпечення ізоляції розділів та керування пам'яттю. Цей рівень не включає стек введення/виведення чи драйверів пристроїв. Саме такий підхід використовує *Hyper-V*. В цій архітектурі стек віртуалізації та драйвери конкретних пристроїв розміщені в спеціальному розділі операційної системи, який має назву кореневого або батьківського розділу. Драйвери працюють в індивідуальному розділі для того, щоб будь-яка гостьова операційна система мала можливість отримати через гіпервізор доступ до обладнання. При цьому кожна віртуальна машина займає цілком відособлений розділ, що позитивно відображається на параметрах захисту та надійності. В мікроядерній моделі гіпервізора (в віртуалізації *Windows Server 2008 R2* використовується саме вона) один розділ являється батьківським (*parent*), а інші – дочірніми (child).

Розділ – це найменша ізольована одиниця, що підтримується гіпервізором. Розмір гіпервізора *Hyper-V* не перевищує 1,5 Мб. Кожному розділу назначаються конкретні апаратні ресурси – частина процесорного часу, об'єм пам'яті, пристрої тощо. Батьківський розділ створює дочірні розділи та керує ними, а також містить стек віртуалізації (*virtualization stack*), що використовується для керування дочірніми розділами. Батьківський розділ створюється першим і володіє всіма ресурсами, які не належать гіпервізору. Володіння всіма апаратними ресурсами означає, що саме кореневий (або батьківський) розділ керує живленням, підключенням пристроїв, що являється самостійно налаштованими, а також керує питаннями апаратних збоїв і навіть завантаженням гіпервізора. В кореневому розділі міститься стек віртуалізації – набір програмних

компонентів, розміщених поверх гіпервізора, який разом з ним забезпечує роботу віртуальних машин.

Мікроядерна модель може дещо програвати монолітній моделі в продуктивності. Але нині пріоритетом стала безпека, тому для більшості компаній цілком прийнятною буде втрата кількох процентів в продуктивності заради скорочення фронту нападів і підвищення стійкості.

Всі версії *Hyper-V* мають один батьківський розділ. Цей розділ керує функціями *Hyper-V*. З кореневого розділу запускається консоль *Windows Server Virtualization*. Крім того, цей розділ використовується для запуску віртуальних машин (*VM*), що підтримують потокову емуляцію старих апаратних засобів. Гостьові *VM* запускаються із дочірніх розділів *Hyper-V*. Архітектура *Hyper-V* наведена на рис. 8.

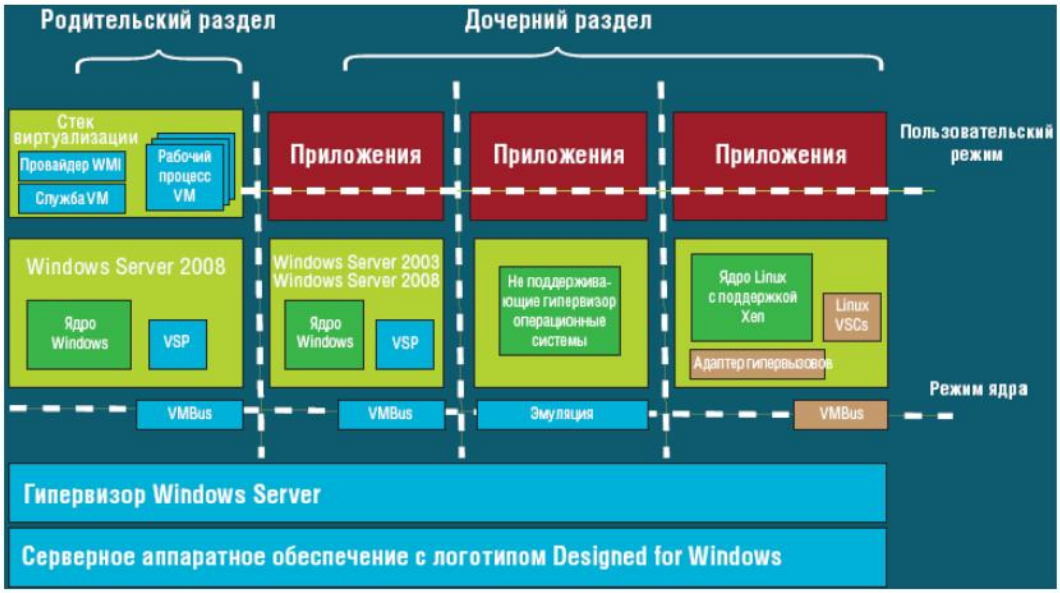

Рис. 8. Архітектура *Hyper-V*.

2. VMware Workstation

В 1998 році *VMware* запатентувала програмні техніки віртуалізації й відтоді випустила немало ефективних програмних забезпечень різного рівня: від *VMware Workstation*, призначеного для настільних ПК до

VMware ESX Server, що дозволяє консолідувати фізичні сервери підприємств у віртуальній інфраструктурі.

На відміну від *ЕОМ* (мейнфреймів), пристрої на базі *x86* не підтримують віртуалізацію в повній мірі. Тому *VMware* прийшлось вирішити немало проблем в процесі створення віртуальних машин для комп'ютерів на базі *x86*. Основні функції більшості *ЦП* (в *ЕОМ* і *ПК*) полягають у виконанні послідовності збережених інструкцій (програм). В процесорах на базі *x86* міститься 17 особливих інструкцій, які створюють проблеми при віртуалізації, і із-за яких операційна система відображає попереджувальне сповіщення, перериває роботу додатку чи просто видає загальний збій. Ці 17 інструкцій стали значною перепоною на початковому етапі впровадження віртуалізації для комп'ютерів на базі *x86*.

Для подолання цієї перешкоди компанія *VMware* розробила адаптивну технологію віртуалізації, яка «перехоплює» дані інструкції на етапі створення і перетворює їх у безпечні інструкції, що є придатними для віртуалізації, не змінюючи при цьому процеси виконання всіх інших інструкцій. В результаті ми отримуємо високопродуктивну віртуальну машину, що відповідає апаратному забезпеченню вузла та підтримує повну програмну сумісність. Компанія *VMware* першою розробила і впровадила дану інноваційну технологію, тому нині вона являється одним із лідерів технологій віртуалізації.

VMware Workstation – платформа, що орієнтована на *desktop*-користувачів і призначена для використання розробниками *ПО*, а також професіоналами в сфері *IT*. Версія популярного продукту *VMware Workstation 7* стала доступна в 2009 році, в якості хостових операційних систем, що підтримується *Windows* та *Linux*. *VMware Workstation 7* може використовуватись разом із середовищем розробки, що робить її особливо популярною серед розробників і спеціалістів технічної підтримки. Вихід

VMware Workstation 7 означає офіційну підтримку *Windows 7* як в якості гостьової, так і хостової операційної системи. Продукт включає підтримку *Aero Peek* та *Flip 3D*, що робить можливим спостерігати за роботою віртуальної машини, наводячи курсор до панелі задач *VMware* чи до відповідної вкладки на робочому столі хосту. Версія може працювати на будь-якій версії *Windows 7*, так само як і будь-які версії *Windows* можуть бути запущені у віртуальних машинах. Крім того, віртуальні машини в *VMware Workstation 7* повністю підтримують *Windows Display Driver Model (WDDM)*, що дозволяє використовувати інтерфейс *Windows Aero* в гостьових машинах.

Рекомендації до виконання

Встановлення та налаштування *Hyper-V*

Необхідні для лабораторної роботи апаратура та програмні інструменти: комп'ютер або сервер, що підтримує віртуалізацію, операційна система *Microsoft Windows Server 2008* (*R2*).

1. Установіть роль *Hyper-V* на сервері *Windows 2008* (можна 2012).

Включіть комп'ютер, дочекайтесь завантаження. Ввійдіть в систему, використовуючи обліковий запис адміністратора. Натисніть *Пуск → Диспетчер серверу → Ролі → Додати ролі*. У відкритому вікні майстра установки виберіть роль серверу *Hyper-V* і виконайте встановлення, виконуючи інструкції майстра (рис. 9).

2. Виконайте мережеві налаштування.

Після успішного встановлення ролі *Hyper-V*, виконайте налаштування віртуальної мережі. Натисніть *Пуск → Диспетчер сервера → Ролі → Hyper-V*. На панелі *Дії* оберіть *Диспетчер віртуальної мережі*. Створіть *Внутрішню* та *Зовнішню* віртуальні мережі.

3. Створення віртуальних машин.

Натисніть *Пуск → Диспетчер сервера → Ролі → Hyper-V.*

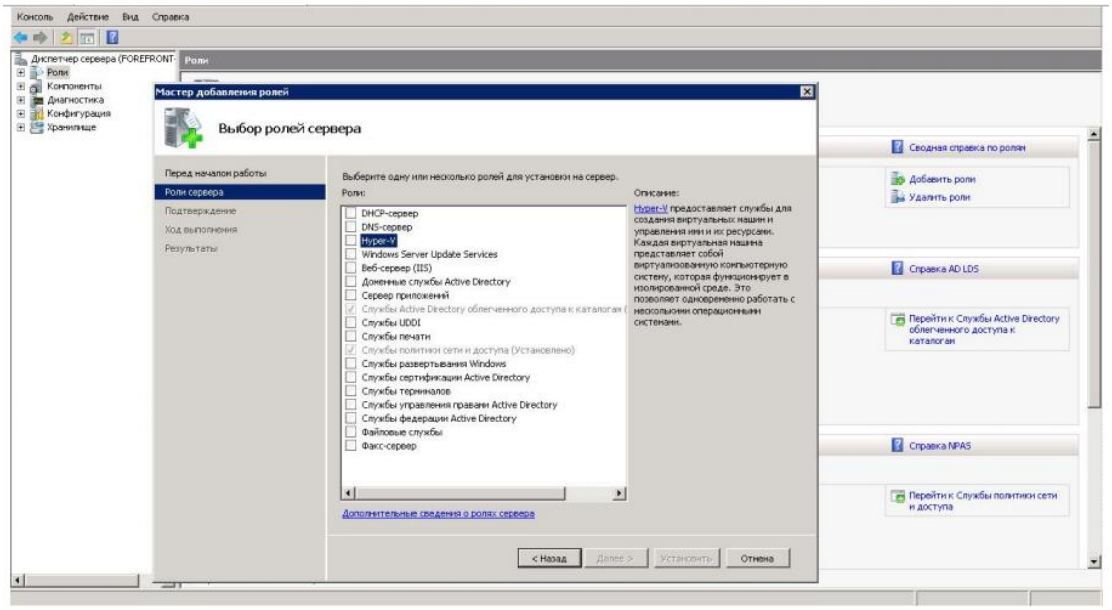

Рис. 9. Встановлення ролі *Hyper-V.*

– Створіть віртуальну машину з фіксованим віртуальним жорстким диском.
– Створіть віртуальну машину з віртуальним жорстким диском, що динамічно розширюється.
– Створіть віртуальну машину для гостьової операційної системи *Windows 7.*
– Змініть кількість оперативної пам’яті, процесорів, тип мережевого підключення в конфігурації віртуальної машини.
– Установіть *Integration Services* для гостьової операційної системи.
– Створіть знімок віртуальної машини.
– Проведіть зміни в гостьовій операційні системі.
– Скасуйте зміни, використовуючи повернення до попереднього знімка.
– Виконайте експорт віртуальної машини.
– Змініть конфігурацію віртуальної машини, змініть розмір диску віртуальної машини.

4. Огляд *System Center Virtual Machine Manager*.

- Запустіть *Configuration Analyzer* для тестування системи.
- Встановіть *SCVMM* сервер.
- Встановіть консоль адміністратора *SCVMM*.
- Додайте користувачів в групу *IT Admin Support*.
- Створіть нову групу вузлів у *VMM*.
- Додайте необхідні сервери *Hyper-V* в *VMM*.
- Змініть конфігурацію віртуальної машини.
- Створіть шаблон віртуальної машини.
- Створіть декілька екземплярів віртуальних машин із отриманого шаблону.
- Зробіть огляд бібліотеки *VMM*.
- Проведіть конвертування фізичного сервера в віртуальне середовище.

Встановлення та налаштування *VMWare Workstation*

Необхідні для лабораторної роботи апаратура та програмні інструменти: настільний або портативний комп'ютер, що підтримує віртуалізацію, операційна система *Microsoft Windows XP, Vista, Windows 7*.

1. Установіть *VMWare Workstation*.

Використовуючи установочний дистрибутив *VMWare Workstation*, установіть продукт на комп'ютер. Запустіть програму, натиснув *Пуск → Всі програми → VMWare → VMWare Workstation* (рис. 10).

2. Виконайте мережеві налаштування.

Натисніть *Пуск → Всі програми → VMWare → Virtual Network Editor* (рис. 11). Виконайте конфігурацію віртуальної мережі.

3. Створіть віртуальну машину для гостьової операційної системи *Windows 7*.

В меню *File* → New → *Virtual Machine* створіть нову віртуальну машину. Встановіть операційну систему *Windows 7* у віртуальній машині.

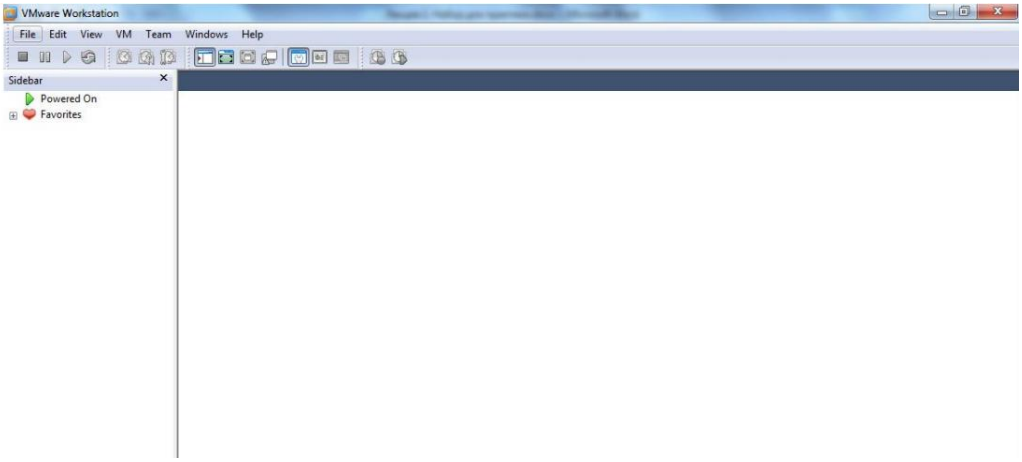

Рис. 10. *VMWare Workstation*.

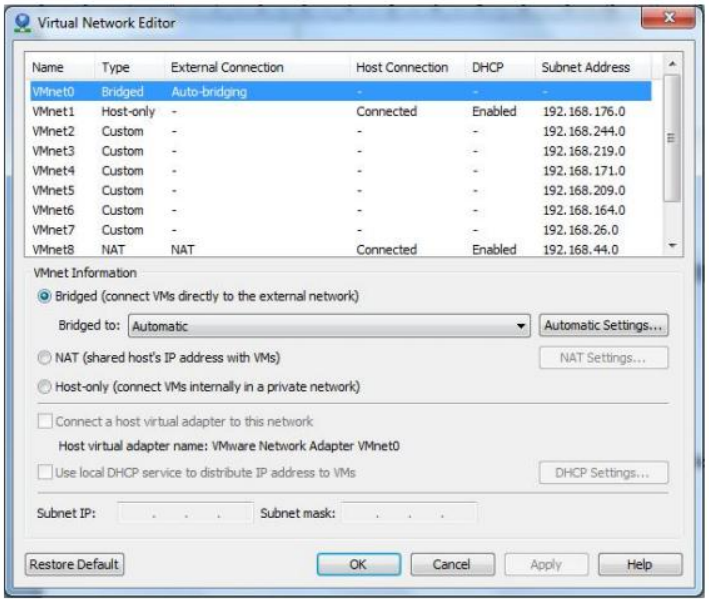

Рис. 11. *Virtual Network Editor*.

4. Встановіть *VMWare Tools*.

Після встановлення операційної системи *Windows 7*, встановіть інструменти *VMWare* в меню VM → *Install VMWare Tools*

5. Створіть знімок віртуальної машини.

Створіть знімок віртуальної машини в меню *VM* → *Snapshot* → *Take Snapshot*.

6. Проведіть зміни в гостьовій операційній системі.

Проведіть довільні зміни в віртуальній машині, скопіюйте на робочий стіл декілька ярликів, створіть декілька папок.

7. Скасуйте зміни, використавши повернення до попереднього знімка.

Виконайте повернення до попереднього знімка віртуальної машини в меню *VM → Snapshot → Revert to Snapshot* та виберіть попередній знімок.

8. Змініть конфігурацію віртуальної машини.

Виконайте зміну конфігурації віртуальної машини в меню *VM → Settings*. Збільште кількість оперативної пам'яті, кількість процесорів, розмір жорсткого диска. Створіть додатковий жорсткий диск (рис. 12).

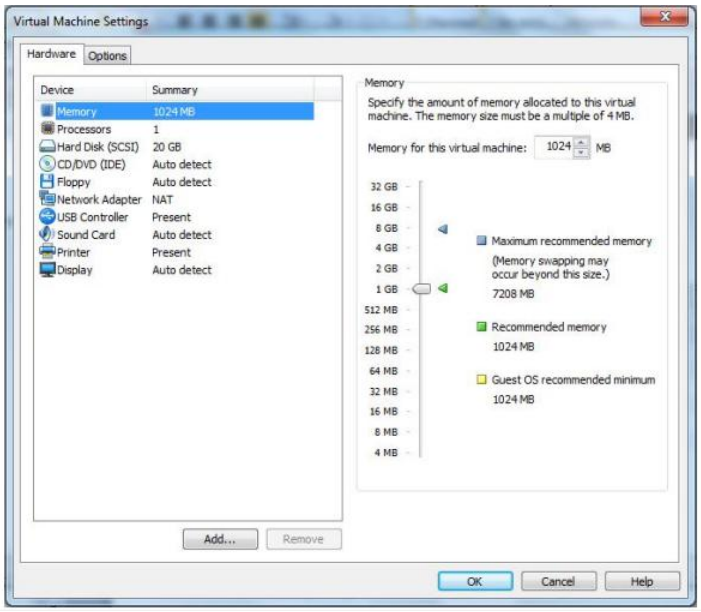

Рис. 12. Налаштування віртуальної машини.

Завдання

Налаштувати віртуальну машину та виконати відповідні налаштування (див. Рекомендації до виконання), використовуючи технології *Microsoft Hyper-V* та *VMware Workstation*.

Вимоги до оформлення звіту

Звіт має включати:

1. Титульний аркуш.

2. Завдання на лабораторну роботу.
3. Результат роботи.

Контрольні запитання

1. Що таке віртуальний комп’ютер? Скільки операційних систем можна встановити на віртуальний комп’ютер?
2. Як створити віртуальний комп'ютер за допомогою технологій *Hyper-V* та *VMware*?
3. Назвіть переваги та недоліки гіпервізора монолітної моделі.
4. Яка архітектура гіпервізора мікроядерної моделі?
5. Що таке розділ гіпервізора? Назвіть види та їх особливості.

Рекомендована література

1. Microsoft 10215A. Implementing and Managing Server Virtualization.
2. Microsoft 6331A. Deploying and Managing Microsoft System Center Virtual Machine Manager.
3. VMware Workstation Pro Documentation [Електронний ресурс]. – Режим доступу: http://www.vmware.com/support/pubs/ws_pubs.html.

ЛАБОРАТОРНА РОБОТА №3. Розробка програми для реальної математичної задачі на кластері НТУУ «КПІ ім. Ігоря Сікорського» [1]

Мета роботи: Навчитись працювати з функціями *MPI* для вирішення прикладних задач.

Завдання: Створити програму відповідно до свого номеру варіанту для вирішення реальної математичної задачі на кластерній системі.

Теоретичні відомості та методичні вказівки

1. Постановка задачі

Двовимірне рівняння Пуассона являє собою диференційне рівняння у часткових похідних виду:

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y).$$

Очевидно, що розв'язати його в аналітичному вигляді немає можливості. Одним із можливих методів розв'язання є застосування методу кінцевих різниць.

2. Метод кінцевих різниць

Метод кінцевих різниць передбачає дискретизацію диференціальних рівнянь на прямокутних координатних сітках. Для двовимірних задач елементарні комірки таких сіток є прямокутниками, а для тривимірних задач комірки становлять паралелепіпеди.

Кінцево-різничні сітки. Розглянемо одновимірну область Θ, що являє собою відрізок $[0 , s]$. Розіб'ємо цей відрізок точками $x_i = ih$, $i = 0, 1, 2, \dots , n$ на n рівних частин довжиною $h = s/n$ кожна. Множина точок $G = \{x_i = ih | i = 0, 1, 2, \dots , n\}$ називається рівномірною одномірною координатною сіткою, а число h – кроком сітки.

Відрізок *[0, s]* можна розбити на *n* частин, вводячи довільні точки $0 < x_1 < x_2 < \ldots < x_{i-1} < x_i < x_{i+1} < \ldots < x_{n-1} < s$.

Координатна сітка $G = \{x_i | i = 0, 1, 2, \ldots, n, x_0 = 0, x_n = s\}$ буде мати крок $h_i = x_i - x_{i-1}$, що залежить від номера i вузла x_i. Якщо $h_i = h_{i+1}$ хоча б для одного номера i, координатна сітка G називається нерівномірною (рис. 13).

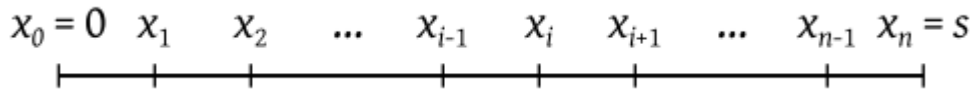

Рис. 13. Одновимірна сітка.

Аналогічно, двовимірною сіткою називають множину точок $G = \{(x_i = ih_1, y_j = jh_2) \mid i = 0, 1, 2, \ldots, n; j = 0, 1, 2, \ldots, m\}$. Рівномірна двовимірна сітка являє собою множину $G = \{(x_i, y_j) \mid i = 0, 1, 2, \ldots, n; j = 0, 1, 2, \ldots, m; x_0 = 0, x_n = s_x, y_0 = 0, y_m = s_y\}$ (рис. 14).

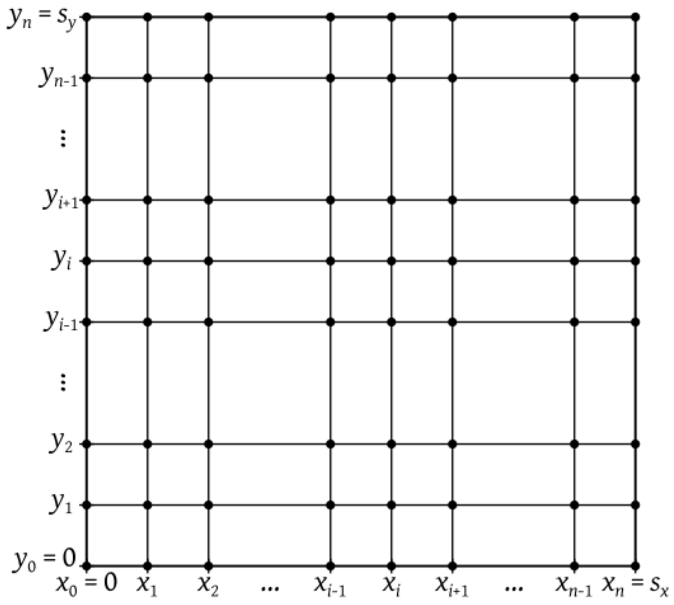

Рис. 14. Двовимірна сітка.

Сіткові функції, кінцеві різниці. Введення для області Θ координатної сітки G передбачає що значення всіх змінних та їх похідних розглядаються тільки у вузлах цієї сітки. З метою виконання цієї умови всі змінні задані сітковими функціями, а похідні будь-якого порядку - кінцевими різницями.

Нехай для деякої області Θ задана сітка $G = \{(x_i, y_j) \mid i = 0, 1, 2, \dots, n; j = 0, 1, 2, \dots, m; x_0 = 0, x_n = s_x, y_0 = 0, y_m = s_y\}$. Тоді функцію $\Phi = \Phi(x_i, y_j)$, $i = 0, 1, 2, \dots, n, j = 0, 1, 2, \dots, m$ дискретного аргументу (x_i, y_j) називають сітковою функцією, визначеною на сітці G.

Будь-якій неперервній функції $f(x, y)$, заданій в області Θ, можна поставити у відповідність сіткову функцію $\Phi(x_i, y_j)$ (для зручності будемо позначати Φ_{ij}), задану на сітці $G = \{(x_i, y_j) \mid i = 0, 1, 2, \dots, n; j = 0, 1, 2, \dots, m; x_0 = 0, x_n = s_x, y_0 = 0, y_m = s_y\}$ (спроектувати функцію $f(x, y)$ на сітку G), беручи до уваги певне правило співвідношення. Наприклад:

$$\Phi_{ij} = f(x_i, y_j)$$

$$\Phi_{ij} = \frac{1}{(x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}})(y_{i+\frac{1}{2}} - y_{i-\frac{1}{2}})} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{i-\frac{1}{2}}}^{y_{i+\frac{1}{2}}} f(x, y)dxdy,$$

де $x_{i\pm\frac{1}{2}}, y_{i\pm\frac{1}{2}}$ – координати серединних точок на відповідних кроках координатної сітки, що визначаються виразами:

$$x_{i+\frac{1}{2}} = \frac{x_{i+1} + x_i}{2}$$

$$x_{i-\frac{1}{2}} = \frac{x_i + x_{i-1}}{2}$$

$$y_{i+\frac{1}{2}} = \frac{y_{i+1} + y_i}{2}$$

$$y_{i-\frac{1}{2}} = \frac{y_i + y_{i-1}}{2}$$

Слід мати на увазі, що одна й та ж сіткова функція, задана на двох різних сітках, що мають спільні вузли, не обов’язково буде мати в цих вузлах рівні значення. За визначенням похідна функції неперервного аргументу x у точці x_0 є ліміт відношення приросту функції до приросту аргументу, коли приріст аргументу близиться до нуля:

$$\frac{\partial f(x)}{\partial x} = \lim_{(x-x_0)\to 0} \frac{f(x) - f(x_0)}{x - x_0}$$

Знехтувавши границею, похідну функції неперервного аргументу можна приблизно замінити (апроксимувати) різницевим виразом, заданим на відповідній сітковій функції $\Phi(x_i,y_j)$. Дана апроксимація може бути виконана декількома способами:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{\Phi_{i+1,j} - \Phi_{i,j}}{\Delta x_i},$$

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{\Phi_{i,j} - \Phi_{i-1,j}}{\Delta x_i},$$

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{\Phi_{i,j+1} - \Phi_{i,j}}{\Delta y_j},$$

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{\Phi_{i,j} - \Phi_{i,j-1}}{\Delta y_j},$$

де $\Delta x_i, \Delta y_j$ – кінцеві різниці координат, що визначаються виразами:

$$\Delta x_i = x_{i+1} - x_i,$$
$$\Delta y_i = y_{j+1} - y_j.$$

Для кожного з перетворень характерна така похибка апроксимації, що прямує до нуля коли крок сітки прямує до нуля.

Похідні другого порядку апроксимуються наступним чином:

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx \frac{\frac{\Phi_{i+1,j} - \Phi_{i,j}}{\Delta x_i} - \frac{\Phi_{i,j} - \Phi_{i-1,j}}{\Delta x_{i-1}}}{\frac{\Delta x_i + \Delta x_{i-1}}{2}}$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} \approx \frac{\frac{\Phi_{i,j+1} - \Phi_{i,j}}{\Delta y_i} - \frac{\Phi_{i,j} - \Phi_{i,j-1}}{\Delta y_{j-1}}}{\frac{\Delta y_i + \Delta y_{i-1}}{2}}$$

У випадку, коли сітка рівномірна вирази спрощуються:

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx \frac{\Phi_{i+1,j} - 2\Phi_{i,j} + \Phi_{i-1,j}}{\Delta x^2}$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} \approx \frac{\Phi_{i,j+1} - 2\Phi_{i,j} + \Phi_{i,j-1}}{\Delta y^2}.$$

3. Аналіз задачі

Застосувавши вище приведений метод до нашої задачі, ми можемо визначити структуру обчислювальної моделі та формули, за якими будемо вести розрахунок. Структура обчислювальної моделі зображена на рис. 15. Сітка дискретизації квадратна та рівномірна. У кожному обчислювальному вузлі буде послідовно розраховуватися матриця точок сітки дискретизації. Реальні вузли будуть працювати паралельно між собою та обмінюватися даними.

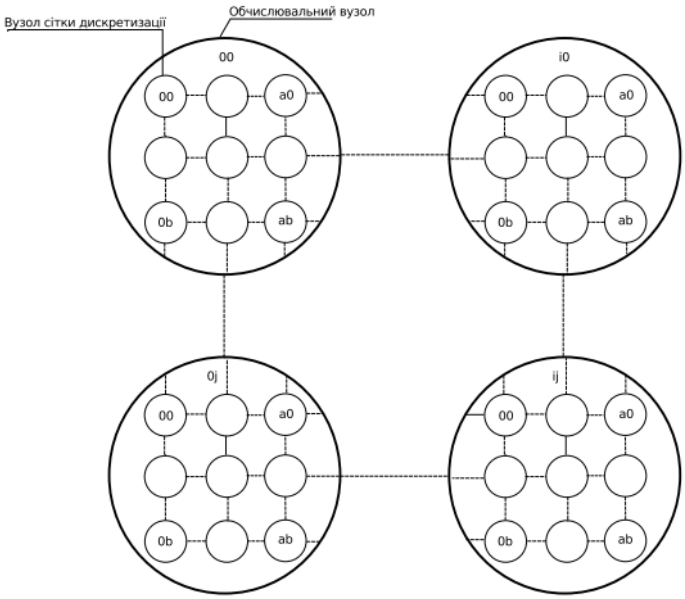

Рис. 15. Структура обчислювальної моделі.

Нехай початкові умови функції всередині області $U_0 = 0$, а на границі області $U_0 = (1 + 3\pi) \sin(3\pi(x + y))$.

$$U_{nm}(z) = U_{nm}(z-1) - \omega\frac{h^2}{4}(LU_{nm} - f_{nm}),$$

де:

- n, m – координати вузла сітки дискретизації;
- z – номер ітерації;

- h – крок у просторі, що визначається за формулою $h = 1/N$;
- N – кількість вузлів сітки дискретизації вздовж однієї з координат;
- f_{nm} – значення правої частини диференційного рівняння у вузлі сітки дискретизації;
- LU_{nm} – різницевий оператор, що визначається за формулою:

$$LU_{nm} = U_{nm} - \frac{U_{n+1,m} - 2U_{n,m} + U_{n-1,m}}{h^2} - \frac{U_{n,m+1} - 2U_{n,m} + U_{n,m-1}}{h^2}.$$

Локальним критерієм завершення алгоритму слугує досягнення заданої точності:

$$\frac{U_{nm}(z) - U_{nm}(z-1)}{U_{nm}(z)} \leq \varepsilon,$$

де ε задається до початку розрахунку.

Рекомендації до виконання

Спрощений алгоритм основного циклу програми виглядає наступним чином:

1. Зробити розрахунки для тих вузлів сітки дискретизації, що не потребують даних від сусідніх процесів.
2. Отримати колонку сітки дискретизації від лівого сусіднього процесу, якщо він є.
3. Передати крайню праву колонку сітки дискретизації до правого сусіднього процесу, якщо він є.
4. Отримати колонку сітки дискретизації від правого сусіднього процесу, якщо він є.
5. Передати крайню ліву колонку сітки дискретизації до лівого сусіднього процесу, якщо він є.
6. Отримати рядок сітки дискретизації від верхнього сусіднього процесу, якщо він є.

7. Передати нижній рядок сітки дискретизації до нижнього сусіднього процесу, якщо він є.
8. Отримати рядок сітки дискретизації від нижнього сусіднього процесу, якщо він є.
9. Передати верхній рядок сітки дискретизації до верхнього сусіднього процесу, якщо він є.
10. Завершити розрахунки для інших вузлів, які не були розраховані у пункті 1.
11. Перевірити локальні критерії завершення алгоритмів. Якщо усі вузли сітки у процесі завершили свою - завершити розрахунки у процесі.
12. Якщо всі процеси завершили розрахунки - завершити алгоритм.

Розглянемо деякі проблеми, зв'язані з реалізацією цього алгоритму:

1) В мові *C* матриці зберігаються по рядках (елементи одного рядку розташовані послідовно в пам'яті). Алгоритм розв'язання задачі передбачає передачу рядків та стовпців матриці. Елементи одного рядку можна передати за допомогою звичайного виклику *MPI_Send*, оскільки вони розташовані послідовно. Елементи одного стовпця таким чином неможливо передати, тому що вони розташовані в пам'яті з кроком, рівним ширині матриці. Для передачі стовпців матриці створимо тип «стовпець матриці» за допомогою функції конструктора типу *MPI_Type_vector* наступним чином:

```
MPI_Type_vector( <висота матриці>,
1, /* довжина одного блоку */
<ширина матриці>,
MPI_DOUBLE,
&column_type);
```

Для передачі стовпця матриці цей тип можна застосувати наступним чином:

```
MPI_Send( <Покажчик на перший елемент стовпця>,
1, /* кількість стовпців */
```

```
column_type,
<ранг задачі- отримувача>,
<тег>,
MPI_COMM_WORLD);
```

2) Аналогічна проблема виникає при початковому розподілі координат вузлів сітки дискретизації по процесах. Фактично нам потрібно відправити квадратну під-матрицю. Реалізація змінюється тільки у ширині блоку і їх кількості при конструюванні типу.

```
MPI_Type_vector( <висота блоку, що відправляємо>,
<ширина блоку, що відправляємо>,
<ширина матриці>,
MPI_DOUBLE,
&new_type);
```

3) Попри те, що процес закінчив розрахунок, він усе одно повинен взаємодіяти з сусідніми процесами, щоб ті теж могли завершити розрахунки. Для цього треба реалізувати подвійну перевірку – умови зупинки розрахунку процесу та умови глобального кінця розрахунку. У програмі спочатку виконується перевірка усіх локальних умов закінчення алгоритму. Якщо усі вузли сітки дискретизації, що належать до процесу, закінчили роботу, то встановлюється ознака закінчення обчислень у процесі. Після цього усі ознаки закінчення обчислень у процесах збираються нульовим процесом і над ними виконується операція логічного «*І*» за допомогою *MPI_Reduce*.

```
MPI_Reduce( &node_stop,
&global_stop,
1,
MPI_SHORT,
MPI_LAND,
0,
MPI_COMM_WORLD);
```

Результат цієї операції є глобальною ознакою закінчення алгоритму, оскільки він матиме значення *true* тільки коли усі процеси закінчили обчислення. Після цього глобальна ознака закінчення обчислення розповсюджується між усіма процесами за допомогою *MPI_Bcast*.

Код програми

Увага! Перед запуском прикладів коду на кластері ознайомтесь з інструкцією роботи на кластері, наведеною в лабораторній роботі №1.

Файл *code/lab_diffeq/main.c*

```
#define _GNU_SOURCE
#include <mpi.h>
#include <math.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdbool.h>
/* Опис типів, структур та функцій */
struct my_matrix /* Структура, що описує матриці */
{
int rows; /* Кількість строк у матриці */
int cols; /* Кількість стовпців у матриці */
double *data; /* Покажчик на матрицю, розвернуту у одномірний масив*/
};
MPI_Datatype node_mat_t; /* Тип для розповсюдження частин сітки
дискретизації по матриці процесів*/
MPI_Datatype mat_col; /* Тип для передачі стовпця матриці */
void inline evaluate( int); /*Функція, що містить ітераційні формули */
double func( double X, double Y); /* Повертає значення правої частини
диференційного рівняння у точці X,Y*/
void init_grid(); /* Ініціалізує значення сітки дискретизації у процесі */
void fatal_error( const char *message, int errorcode); /* Функція виведення
помилок вводу-виводу */
struct my_matrix *matrix_alloc( int rows, int cols, double initial); /*
Функція для виділення пам’яті та ініціалізацію матриці*/
void matrix_print( const char *filename, struct my_matrix *mat); /* Функція
для виведення матриці у файл*/
struct my_matrix *read_matrix( const char *filename); /* Функція
для зчитування матриці із файлу*/
/* ********************************************************************
*/
/* Опис змінних */
int np; /* Кількість процесів */
int rank; /* Ранг процесу у MPI_COMM_WORLD */
int nodes_width; /* Ширина матриці процесів */
int node_X; /* Координата X у матриці процесів */
int node_Y; /* Координата Y у матриці процесів */
int total_width; /* Ширина матриці вузлів сітки дискретизації*/
int points_per_node; /* Ширина частини матриці вузлів сітки дискретизації,
що припадає на кожен процес*/
const char* inpfileX=”inpX.csv”; /* Ім’я файлу з якого вводяться
горизонтальні координати сітки дискретизації */
const char* inpfileY=”inpY.csv”; /* Ім’я файлу з якого вводяться вертикальні
координати сітки дискретизації */
const char* inpfileInit=”inpInit.csv”; /*Ім’я файлу з якого вводяться
початкові та граничні умови для сітки дискретизації*/
const char* outpfile=”outp.csv”; /* Ім’я файлу куди виводиться результат */
my_matrix* allcoords_X; /* Матриця реальних горизонтальних координат вузлів
```

```
сітки дискретизації */
my_matrix* allcoords_Y; /* Матриця реальних вертикальних координат вузлів
сітки дискретизації */
my_matrix* all_grid_init; /*Всі початкові та граничні умови функції*/
my_matrix* total_mat; /* Повна сітка дискретизації, що виводиться у процесі
0*/
my_matrix* grid_init; /*Частина початкових умов для процесу*/
my_matrix* coords_X; /* Частина матриці реальних координат сітки
дискретизації, яка зберігається у кожному з процесів*/
my_matrix* coords_Y; /* Частина матриці реальних координат сітки
дискретизації, яка зберігається у кожному з процесів*/
my_matrix* node_mat; /* Частина сітки дискретизації, яка зберігається у
кожному з процесів*/
my_matrix* f_xy; /* Масив значень функції в вузлах сітки дискретизації*/:
MPI_Status stat; /* Змінна, у яку повертається статус прийому повідомлень */
double epsilon=0.01; /* Точність рішення*/
double omega=0.4; /* Коефіцієнт релаксації*/
int max_iter=1000000; /*Максимальна кількість ітерацій у випадку нев'язки
функції */
double h; /* Крок сітки дискретизації */
bool* local_stop; /* Масив для запам'ятовування локальних умов зупинки */
int* node_iter;
short node_stop; /* Ознака завершення обчислень у процесі*/
short global_stop; /* Глобальна ознака завершення обчислень*/
double* left_col; /* Стовпці, що процес буде приймати з лівого та правого
процесів під час ітерацій */
double* right_col;
double* top_row; /* Рядки, що процес буде приймати з верхнього та нижнього
процесів під час ітерацій */
double* bot_row;
/* ********************************************************************
*/
 /* Головна програма */
int main( int argc, char *argv[])
{
MPI_Init( &argc, &argv); /* Ініціалізуємо середовище */
MPI_Comm_size( MPI_COMM_WORLD, &np); /* Отримуємо розмір MPI_COMM_WORLD */
MPI_Comm_rank( MPI_COMM_WORLD, &rank); /* Отримуємо ранг процесу у
MPI_COMM_WORLD*/
nodes_width=sqrt(np); /* Отримуємо ширину матриці процесів*/
node_X=rank%nodes_width; /* Отримуємо горизонтальну координату процесу у
матриці */
node_Y=rank/nodes_width; /* Отримуємо вертикальну координату процесу у
матриці */
if (rank==0)
{
allcoords_X=read_matrix(inpfileX); /* Вводимо координати */
allcoords_Y=read_matrix(inpfileY); /* Вводимо координати */
all_grid_init=read_matrix(inpfileInit); /* Вводимо координати */
total_width=allcoords_X- >rows; /* Отримуємо ширину сітки дискретизації */
}
MPI_Bcast( &total_width, 1, MPI_INT, 0, MPI_COMM_WORLD); /* Розповсюджуємо
ширину
сітки дискретизації */
h=1.0/total_width; /* Визначаємо крок сітки дискретизації */
points_per_node=total_width/nodes_width;
/* Створюємо тип, що відповідає частині матриці, яка буде передаватися */
```

```
MPI_Type_vector(points_per_node,points_per_node,total_width, MPI_DOUBLE,
&node_mat_t);
MPI_Type_commit( &node_mat_t); /* Реєструємо тип */
coords_X=matrix_alloc(points_per_node,points_per_node, 0.0);
coords_Y=matrix_alloc(points_per_node,points_per_node, 0.0);
grid_init=matrix_alloc(points_per_node,points_per_node, 0.0);
if(rank==0)
{/* Процес 0 розсилає реальні координати сітки дискретизації*/
for( int i=0;i<nodes_width;i++)
{
for( int j =0;j <nodes_width;j ++)
{
if( ! (i==0&&j ==0))
{
MPI_Send(allcoords_X- >data+i*total_width*points_per_node+j *points_per_node
, 1,node_mat_t,i*nodes_width+j, 0, MPI_COMM_WORLD);
MPI_Send(allcoords_Y- >data+i*total_width*points_per_node+j *points_per_node
, 1,node_mat_t,i*nodes_width+j, 0, MPI_COMM_WORLD);
MPI_Send(all_grid_init- >data+i*total_width*points_per_node+j
*points_per_node
, 1,node_mat_t,i*nodes_width+j, 0, MPI_COMM_WORLD);
}
}
}
for( int i=0;i<points_per_node;i++)
{
for( int j =0;j <points_per_node;j ++)
{
coords_X- >data[i*points_per_node+j] =allcoords_X- >data[i*total_width+j];
 coords_Y- >data[i*points_per_node+j] =allcoords_Y- >data[i*total_width+j];
grid_init- >data[i*points_per_node+j] =all_grid_init- >data[i*total_width+j];
}
}
}
else
{/* Інші процеси їх приймають */
my_matrix* temp_X=matrix_alloc(total_width,points_per_node, 0);
my_matrix* temp_Y=matrix_alloc(total_width,points_per_node, 0);
my_matrix* temp_init=matrix_alloc(total_width,points_per_node, 0);
MPI_Recv(temp_X- >data, 1,node_mat_t, 0, 0, MPI_COMM_WORLD, &stat);
MPI_Recv(temp_Y- >data, 1,node_mat_t, 0, 0, MPI_COMM_WORLD, &stat);
MPI_Recv(temp_init- >data, 1,node_mat_t, 0, 0, MPI_COMM_WORLD, &stat);
for( int i=0;i<points_per_node;i++)
{
for( int j =0;j <points_per_node;j ++)
{
coords_X- >data[i*points_per_node+j] =temp_X- >data[i*total_width+j];
coords_Y- >data[i*points_per_node+j] =temp_Y- >data[i*total_width+j];
grid_init- >data[i*points_per_node+j] =temp_init- >data[i*total_width+j];
}
}
}
/* Ініціалізуємо умови локальної зупинки алгоритму*/
local_stop=(bool*)malloc(points_per_node*points_per_node*sizeof(bool));
for( int i=0;i<points_per_node*points_per_node;i++)
{
local_stop[i] =false;
```

```
}
for( int i=0;i<points_per_node;i++) /* Граничні точки залишаються
постійними*/
{
if(node_Y==0)
{
local_stop[i] =true;
}
if(node_X==0)
{
local_stop[i*points_per_node] =true;
}
if(node_X==nodes_width- 1)
{
local_stop[(i+1) *points_per_node- 1] =true;
}
if(node_Y==nodes_width- 1)
{
local_stop[points_per_node*(points_per_node- 1) +i] =true;
}
}
/* Ініціалізуємо і встановлюємо початкові значення внутрішнього діапазону */
node_mat=matrix_alloc(points_per_node,points_per_node, 0);
/* Встановлюємо початкові значення граничних вузлів сітки дискретизації */
for( int i=0;i<points_per_node*points_per_node;i++)
{
node_mat- >data[i] =grid_init- >data[i];
}
/* Створимо масив значень функції в локальних вузлах сітки дискретизації*/
f_xy=matrix_alloc(points_per_node,points_per_node, 0.0);
for( int i=0;i<points_per_node*points_per_node;i++)
{
f_xy- >data[i] =func(coords_X- >data[i],coords_Y- >data[i]);
}
/*Визначимо типи для передач під час ітерацій*/
/* Тип, що визначає стовпець матриці*/
MPI_Type_vector(points_per_node, 1,points_per_node, MPI_DOUBLE, &mat_col);
MPI_Type_commit( &mat_col); /* Реєструємо тип */
left_col=( double*)malloc(points_per_node*points_per_node*sizeof( double));
right_col=( double*)malloc(points_per_node*points_per_node*sizeof( double));
top_row=( double*)malloc(points_per_node*points_per_node*sizeof( double));
bot_row=( double*)malloc(points_per_node*points_per_node*sizeof( double));
node_iter=( int*)malloc(points_per_node*points_per_node*sizeof( int));
for( int i=0;i<points_per_node*points_per_node;i++)
{
node_iter[i] =0;
}
/* Початок ітерацій */
do
{
/* Обмін між процесами у сітці*/
if(node_X! =0)
{
/* Приймемо стовпець з лівого процесу */
MPI_Recv(left_col, 1,mat_col,rank- 1,rank- 1, MPI_COMM_WORLD, &stat);
}
if(node_X! =nodes_width- 1)
```

```
{
/* Відішлемо стовпець до правого процесу */
MPI_Send(node_mat- >data+points_per_node- 1, 1,mat_col
,rank+1,rank, MPI_COMM_WORLD);
/* Приймемо стовпець з правого процесу */
MPI_Recv(right_col, 1,mat_col,rank+1,rank+1, MPI_COMM_WORLD, &stat);
}
if(node_X! =0)
{
 /* Відішлемо стовпець до лівого процесу */
MPI_Send(node_mat- >data, 1,mat_col,rank- 1,rank, MPI_COMM_WORLD);
}
if(node_Y! =0)
{
/* Приймемо строку з верхнього процесу */
MPI_Recv(top_row,points_per_node, MPI_DOUBLE,rank- nodes_width
,rank- nodes_width, MPI_COMM_WORLD, &stat);
}
if(node_Y! =nodes_width- 1)
{
/* Відішлемо строку до нижнього процесу */
MPI_Send(node_mat- >data+(points_per_node- 1)
*points_per_node,points_per_node
, MPI_DOUBLE,rank+nodes_width,rank, MPI_COMM_WORLD);
/* Приймемо строку з нижнього процесу */
MPI_Recv(bot_row,points_per_node, MPI_DOUBLE,rank+nodes_width,
rank+nodes_width, MPI_COMM_WORLD, &stat);
}
if(node_Y! =0)
{
/* Відішлемо строку до верхнього процесу */
MPI_Send(node_mat- >data,points_per_node, MPI_DOUBLE,
rank- nodes_width,rank, MPI_COMM_WORLD);
}
/* Виклик функції, що проводить ітераційну обробку*/
for( int i=0;i<points_per_node;i++)
{
for( int j =0;j <points_per_node;j ++)
{
evaluate(i*points_per_node+j);
}
}
/*Якщо усі вузли сітки дискретизації у процесі завершили
обчислення - встановлюємо ознаку завершення обчислень у процесі*/
node_stop=1;
for( int i=0;i<points_per_node*points_per_node;i++){
if ( ! local_stop[i]){
node_stop=0;
break;
}
}
global_stop=1;
/*Перевіряємо чи всі процеси закінчили обчислення*/
MPI_Reduce( &node_stop, &global_stop, 1, MPI_SHORT, MPI_BAND, 0,
MPI_COMM_WORLD);
 /*Якщо всі - то посилаємо сигнал про завершення*/
MPI_Bcast( &global_stop, 1, MPI_SHORT, 0, MPI_COMM_WORLD);
```

```
} while( ! global_stop);
/* Збираємо результат */
total_mat=matrix_alloc(total_width,total_width, 0.0);
if(rank==0){
for( int i=0;i<nodes_width;i++){
for( int j =0;j <nodes_width;j ++){
if( ! (i==0&&j ==0)){
my_matrix* temp=matrix_alloc(points_per_node,points_per_node, 0.0);
/* Приймаємо частини матриці сітки дискретизації з кожного процесу */
MPI_Recv(temp- >data,points_per_node*points_per_node,
MPI_DOUBLE,i*nodes_width+j,i*nodes_width+j, MPI_COMM_WORLD, &stat);
/* Та розташовуємо їх у повній матриці */
for( int k=0;k<points_per_node;k++){
for( int l=0;l<points_per_node;l++){
total_mat- >data[i*total_width*points_per_node+
j *points_per_node+k*total_width+l]
=temp- >data[k*points_per_node+l];
}
}
}
}
}
for( int i=0;i<points_per_node;i++){
for( int j =0;j <points_per_node;j ++){
total_mat- >data[i*total_width+j] =node_mat- >data[i*points_per_node+j];
}
}
}
else{
MPI_Send(node_mat- >data,points_per_node*points_per_node
, MPI_DOUBLE, 0,rank, MPI_COMM_WORLD);
}
if (rank==0){
matrix_print(outpfile,total_mat);;
}
MPI_Finalize();
}
/* ************************************************************************
*/
/* Допоміжні функції */
double func( double X, double Y)
{
return X*Y;
}
/*Функція, що містить ітераційні формули */
 void inline evaluate( int index)
{
/*Якщо досягнута потрібна точність,
то не потрібно проводити обчислення */
if (local_stop[index])
{
return;
}
/*Якщо вузол сітки знаходиться на границі між обчислювальними
вузлами, то потрібно взяти значення з буфера передачі*/
double left=index%points_per_node==0
?left_col[index] : node_mat- >data[index - 1];
```

```
double right=(index + 1) %points_per_node==0
?right_col[index+1- points_per_node] : node_mat- >data[index + 1];
double top=index/points_per_node == 0 ? top_row[index % points_per_node]
: node_mat- >data[index- points_per_node];
double bottom=index/points_per_node==points_per_node- 1
?bot_row[index%points_per_node]
: node_mat- >data[index+points_per_node];
/*Власне обчислення*/
double old_node=node_mat- >data[index];
double LU=old_node- (left+right+top+bottom- 4*old_node) /(h*h);
node_mat- >data[index] =old_node- omega*h*h/4*(LU- f_xy- >data[index]);
/*Якщо досягнута точність або перевищена максимальна кількість ітерацій,
то завершуємо обчислення у цьому вузлі сітки*/
if (fabs(node_mat- >data[index] - old_node) <=
fabs(node_mat- >data[index] *epsilon) | | ++node_iter[index] >max_iter)
{
local_stop[index] =true;
}
}
/* Ініціалізує значення сітки дискретизації у процесі */
void init_grid()
{
for( int i=0;i<points_per_node*points_per_node;i++)
{
node_mat- >data[i] =grid_init- >data[i];
}
}
void fatal_error( const char *message, int errorcode)
{
printf( "fatal error: code %d, %s\n", errorcode, message);
fflush(stdout);
MPI_Abort( MPI_COMM_WORLD, errorcode);
}
struct my_matrix *matrix_alloc( int rows, int cols, double initial)
{
struct my_matrix *result = (my_matrix*)malloc( sizeof( struct my_matrix));
result- >rows = rows;
 result- >cols = cols;
result- >data = ( double*)malloc( sizeof( double) * rows * cols);
for( int i = 0; i < rows; i++)
{
for( int j = 0; j < cols; j ++)
{
result- >data[i * cols + j] = initial;
}
}
return result;
}
void matrix_print( const char *filename, struct my_matrix *mat)
{
FILE *f = fopen(filename, "w");
if(f == NULL)
{
fatal_error( "cant write to file", 2);
}
for( int i = 0; i < mat- >rows; i++)
{
```

```
for( int j = 0; j < mat- >cols; j ++)
{
fprintf(f, ”%lf ”, mat- >data[i * mat- >cols + j]);
}
fprintf(f, ”\n”);
}
fclose(f);
}
struct my_matrix *read_matrix( const char *filename)
{
FILE *mat_file = fopen(filename, ”r”);
if(mat_file == NULL)
{
fatal_error( ”can’t open matrix file”, 1);
}
int rows;
int cols;
fscanf(mat_file, ”%d %d”, &rows, &cols);
struct my_matrix *result = matrix_alloc(rows, cols, 0.0);
for( int i = 0; i < rows; i++)
{
for( int j = 0; j < cols; j ++)
{
fscanf(mat_file, ”%lf”, &result- >data[i * cols + j]);
}
}
fclose(mat_file);
return result;
}
```

Завдання

Створити програму відповідно до свого номеру варіанту для вирішення наведеної задачі на кластерній системі.

Варіанти завдань

Варіант завдання	Тип пересилок	Тип простору	Тип сітки
1	Блокуючі	Двовимірний	Однорідна
2	Неблокуючі	Двовимірний	Однорідна
3	Блокуючі	Двовимірний	Неоднорідна
4	Неблокуючі	Двовимірний	Неоднорідна
5	Блокуючі	Тривимірний	Однорідна

6	Неблокуючі	Тривимірний	Однорідна
7	Блокуючі	Тривимірний	Неоднорідна
8	Неблокуючі	Тривимірний	Неоднорідна

Контрольні запитання

1. Які базові функції *MPI* ви знаєте?
2. Для чого використовується функція *MPI_Send*?
3. Для чого використовується функція конструктора типу *MPI_Type_vector*?
4. Для чого використовується функція *MPI_Reduce*?
5. Для чого використовується функція *MPI_Bcast*?

Рекомендована література

1. Засоби паралельного програмування [Текст] / С.Г. Стіренко, Д.В. Грибенко, О.І. Зіненко, А.В. Михайленко– К., 2011. – 154 с.
2. Жуков І.А. Паралельні та розподілені обчислення [Текст] / І.А. Жуков, О.В. Корочкін. – Київ: «Корнійчук», 2014. – 284 с.